Advanced Programming

Frank Filthaut

Scope

Course aims: students with some Python programming skills but little beyond this

Course coverage:

- C++ syntax & basics
- language binding
- multi-threading

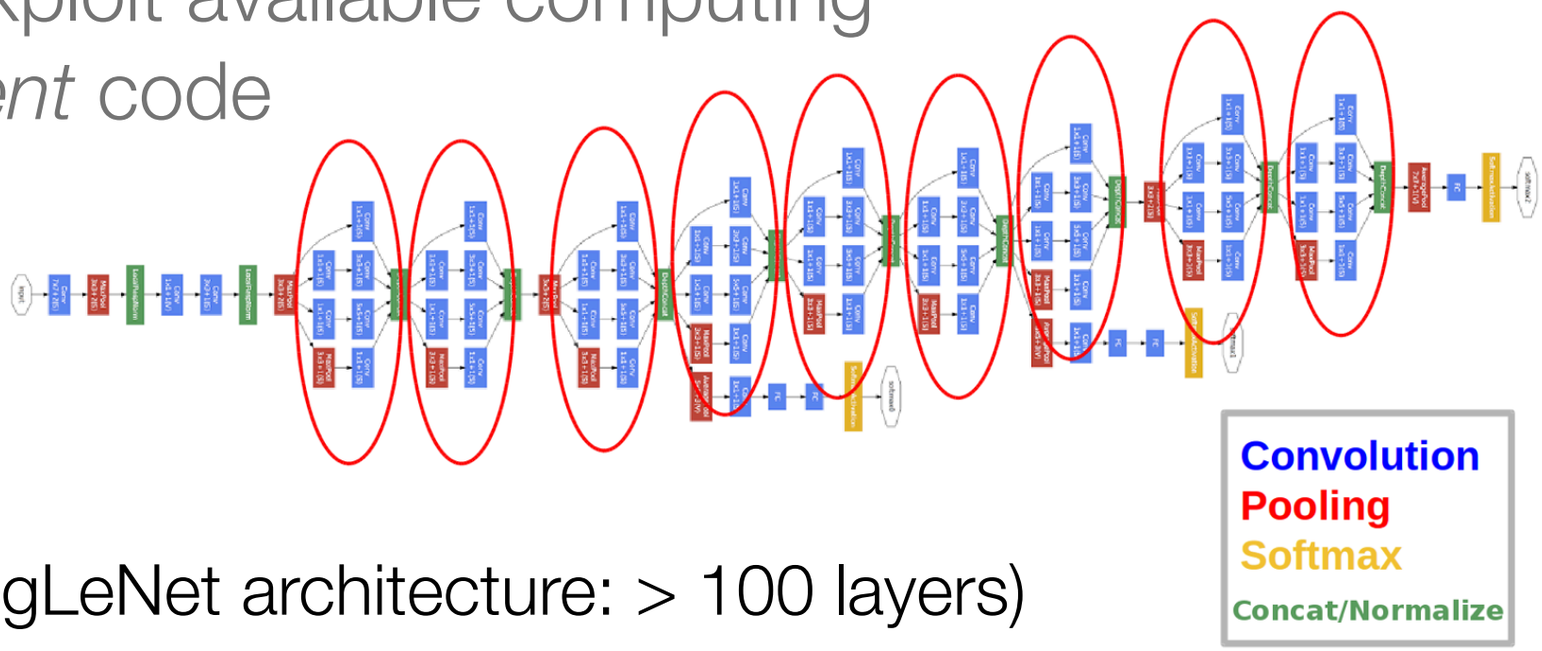
Machine Learning

Earliest incarnations (back-propagation, few layers, limited #nodes per layer) were “simple” but not very powerful

Wealth of applications now in existence (e.g. unsupervised learning for feature recognition) are only possible thanks to great advances in computing power (Moore’s law)

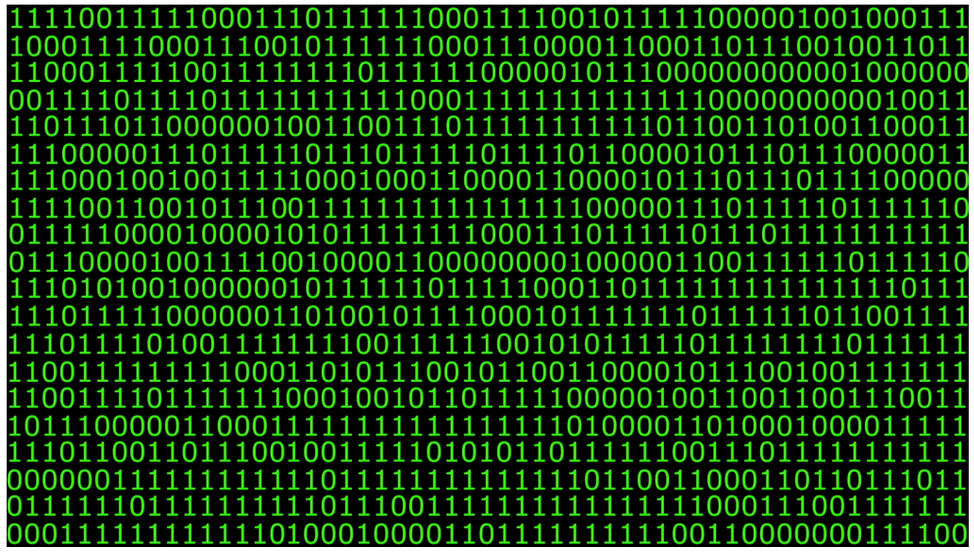
- millions of network parameters to be optimised

In order to optimally exploit available computing resources, need *efficient* code



(GoogLeNet architecture: > 100 layers)

Compiled versus interpreted languages



```
def eval_AtA_times_u(u):
    return eval_At_times_u(eval_A_times_u(u))


def main():
    n = int(argv[1])
    u = [1] * n
    local_eval_AtA_times_u = eval_AtA_times_u

    for dummy in range(10):
        v = local_eval_AtA_times_u(u)
        u = local_eval_AtA_times_u(v)

    vBv = vv = 0

    for ue, ve in zip(u, v):
        vBv += ue * ve
        vv  += ve * ve

    print("%0.9f" % (sqrt(vBv/vv)))

main()
```

- binary code: directly executable on specific architecture: fast but not portable
- interpreted code: portable but significantly slower, as code needs to be interpreted at run time (and multiple times, in case of loops)
  - Python does convert to .pyc behind the scenes, but this is not to binary code

Practicalities: lectures

The best way to learn to “speak” a programming language is as for natural languages: by doing rather than by listening

Intention: minimise time spent in lectures

- expect one or a few more actual lectures, on specialised topics
  - towards the end of the course
- otherwise, will only have computer tutorial sessions
  - apologies: I requested 2 blocks of 2 hours per week but got only 1, and this cannot be fixed anymore in a way that accommodates all
  - 1st official tutorial session next week; assistance by Edwin Chow. Will determine how to proceed from there

Practicalities: exercises/projects

You will pass the “exam” (no grade) upon handing in exercises to satisfaction

- we will start with simple exercises; expectation is that towards the end, some of these will develop into a somewhat larger project

Exercises will mostly follow those from another course (see separate material), but with modifications

- different choice of material (also in view of previous point)
- different way to hand in results

You are welcome to use an IDE for your code development (as long as you submit source code that works with straight g++)

- geany, gedit, kate, **eclipse**, atom all appear to be available on the faculty’s Linux systems

You are welcome to inspect code performance (or test other aspects) using valgrind (also available)

Contents

As discussed in later sections (separate documents):

Proposed approach

Material on the previous slides is rather a lot; strive to see where corners can be cut

- object orientation? (Python uses OO as well)
- exception handling

Would help to know your background / experience

- will adapt selection of material / exercises (within bounds)